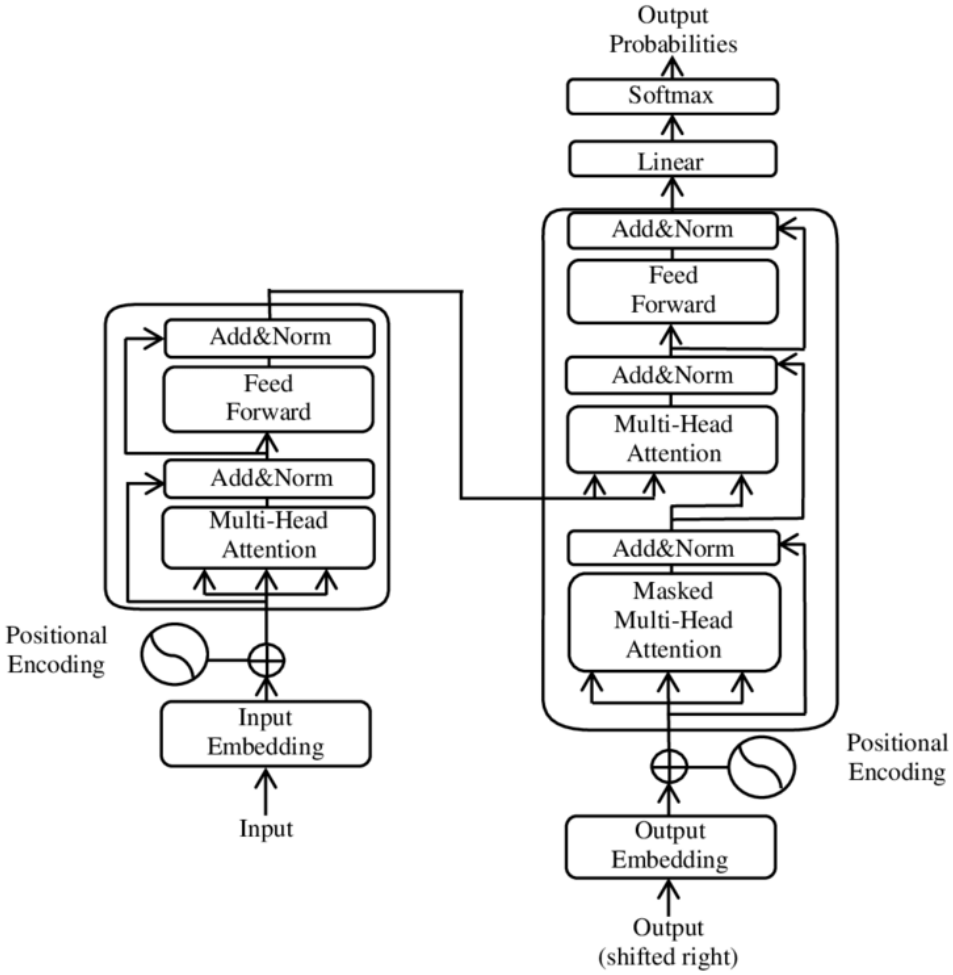
Lecture 13: Transformers

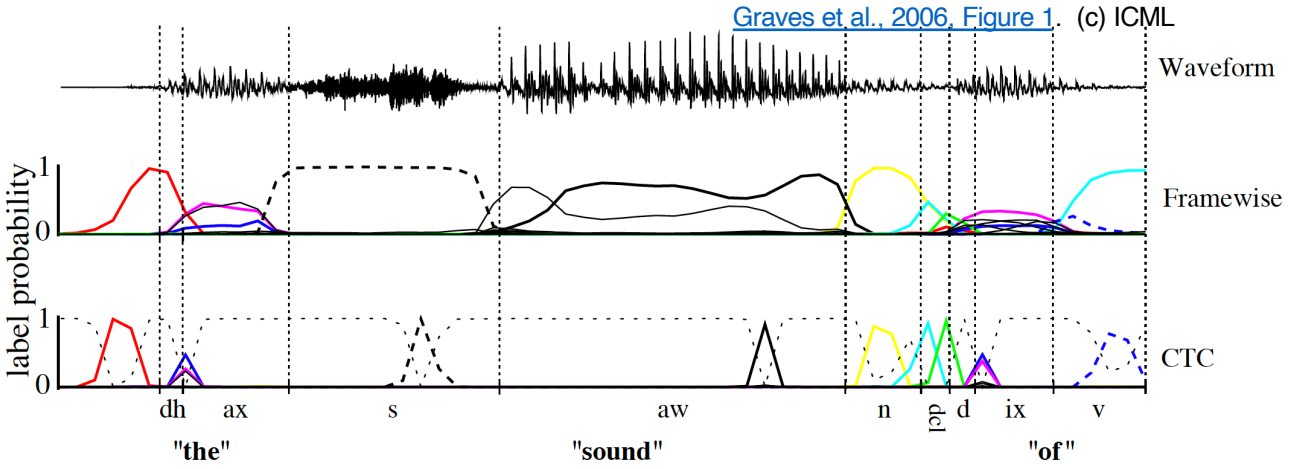
Mark Hasegawa-Johnson

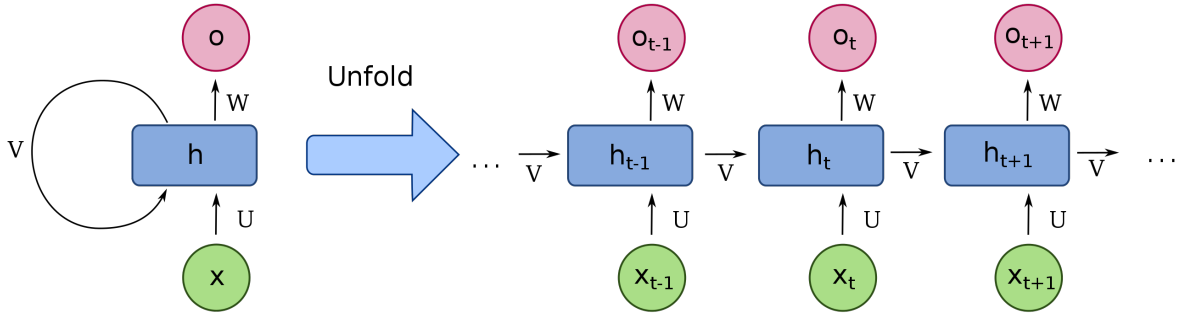


Outline

- Recurrent neural networks
- Attention
- Positional embedding

Recurrent neural network

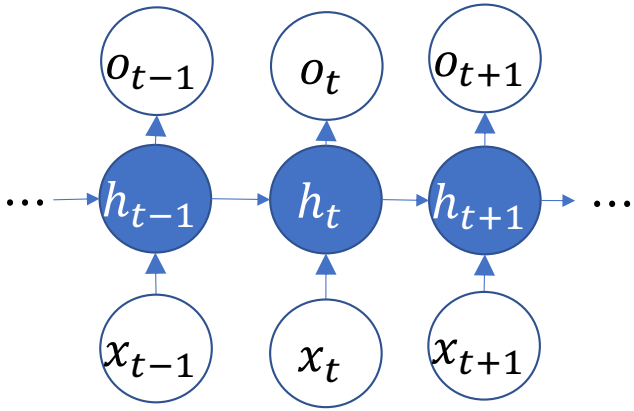




By fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60109157

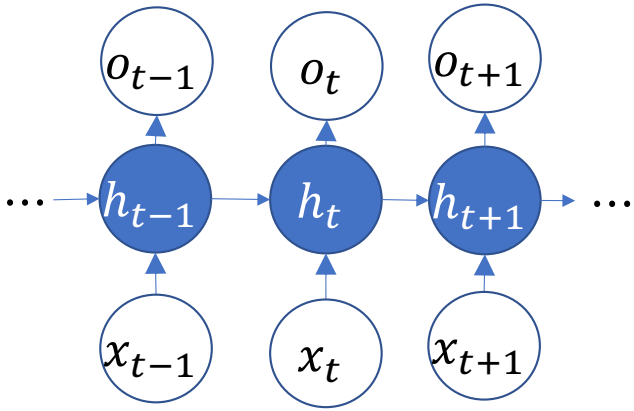
- In a recurrent neural network (RNN), the hidden node activation vector, h_t, depends on the value of the same vector at time $t-1$.
- From 2014-2017, the best speech recognition and machine translation used RNNs.
- The input is x_t=speech or input-language text
- The output is o_t=text in the target language

Example: Part of speech tagging



- $\boldsymbol{x}_t$ =vector representation of the t[th] word
- $\boldsymbol{h}_t$ =hidden state vector = $\tanh(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$
- $\boldsymbol{o}_t = \text{softmax}(\boldsymbol{W}\boldsymbol{h}_t) = [P(Y_t = \text{Noun}|X_1, \dots, X_t), P(Y_t = \text{Verb}|X_1, \dots, X_t), \dots]$

Training an RNN



An RNN is trained using gradient descent, just like any other neural network!

$$u_{j,i} \leftarrow u_{j,i} - \eta \frac{\partial \mathfrak{L}}{\partial u_{j,i}}$$

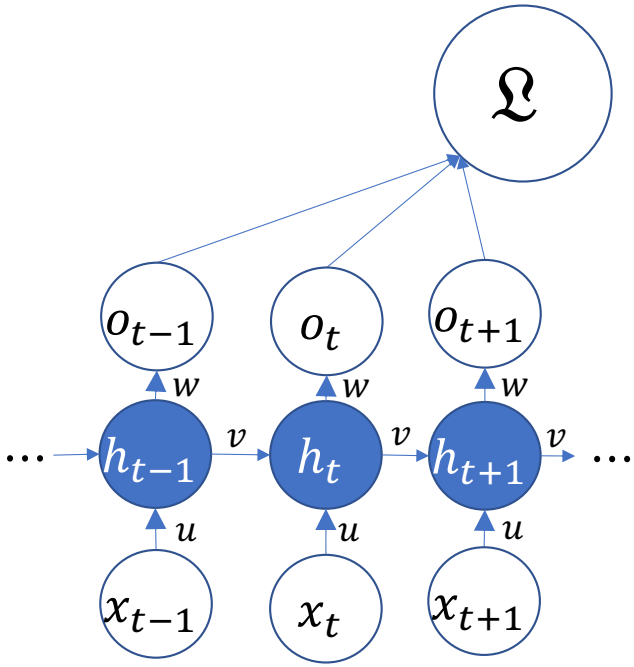
$$w_{j,k} \leftarrow w_{j,k} - \eta \frac{\partial \mathfrak{L}}{\partial w_{j,k}}$$

…where $\mathfrak{L}$ is the loss function, and η is a step size.

Training an RNN: Infinite recursion?

The big difference is that now the loss function depends on $\boldsymbol{U}, V$ and $\boldsymbol{W}$ in many different ways:

- The loss function depends on each of the state vectors $\boldsymbol{h}_t$, which depends directly on $\boldsymbol{U}$ and $\boldsymbol{V}$.
- But $\boldsymbol{h}_t$ also depends on $\boldsymbol{h}_{t-1}$, which, in turn, depends on $\boldsymbol{U}$ and $\boldsymbol{V}$.
- … and so on.

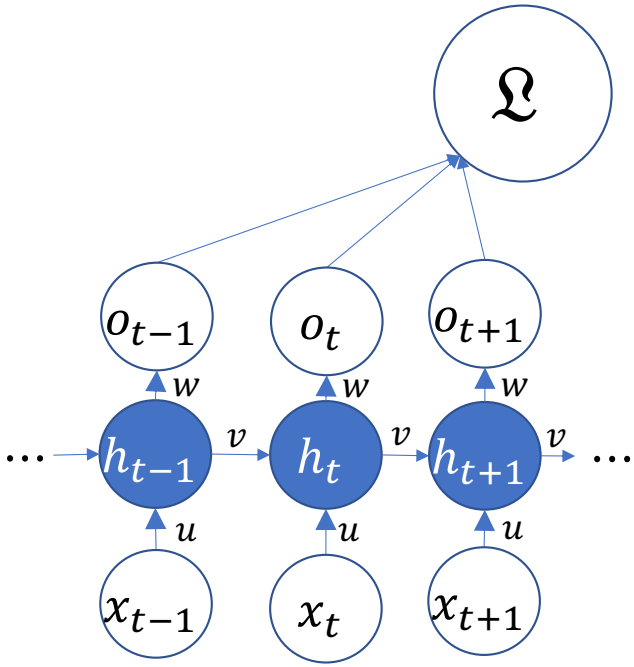


Back-propagation through time

The solution is something called back-propagation through time:

$$\frac{d\mathfrak{L}}{dh_{i,t}} = \frac{\partial \mathfrak{L}}{\partial h_{i,t}} + \sum_j \frac{d\mathfrak{L}}{dh_{j,t+1}} \frac{\partial h_{j,t+1}}{\partial h_{i,t}}$$

- The first term measures losses caused directly by $h_{i,t}$, for example, if $o_{i,t}$ is wrong.
- The second term measures losses caused indirectly, for example, because $h_{i,t}$ caused $h_{j,t+1}$ to be wrong.

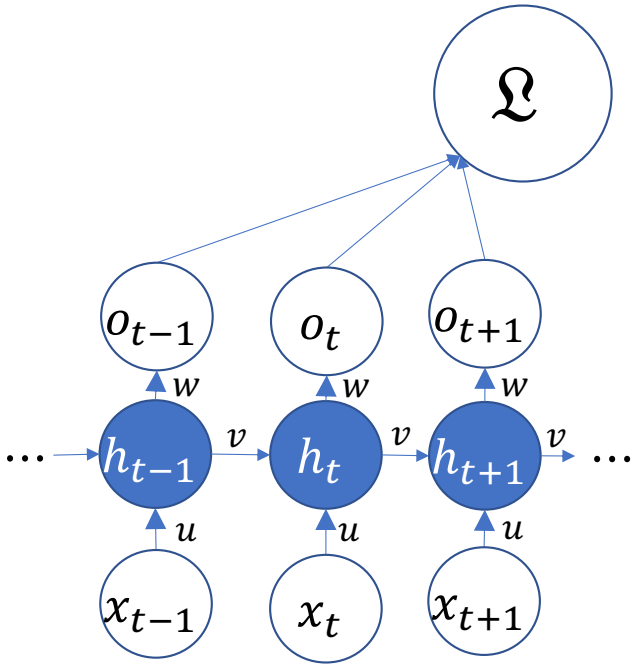


Back-propagation through time

Notice that this is just like training a very deep network!

- Back-propagation through time: back-propagate from time step $t + 1$ to time step t
- Back-propagation in a very deep network: back-propagate from layer $l + 1$ to layer l

Toolkits like PyTorch may use the same code in both cases.



Outline

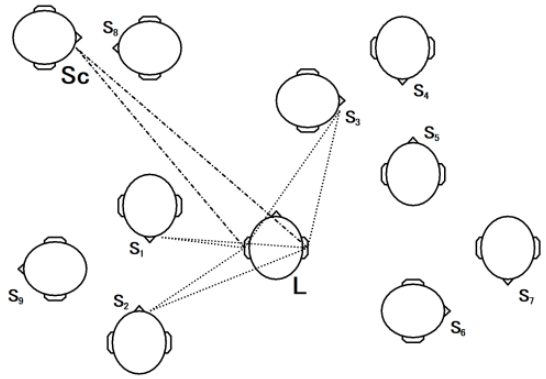
- Recurrent neural networks
- Attention
- Positional embedding

The Cocktail-Party Effect

- If you are focusing on one person's voice, but hear your name spoken by another person, your attention immediately shifts to the second voice.
- This "cocktail-party effect" suggests a model of hearing in which all sounds are processed preconsciously. Trigger sounds in an unattended source will cause attention to re-orient to that source.

https://commons.wikimedia.org/wiki/File:Cocktail_party_attendees_at_Fuller_Lodge,_1946.jpg

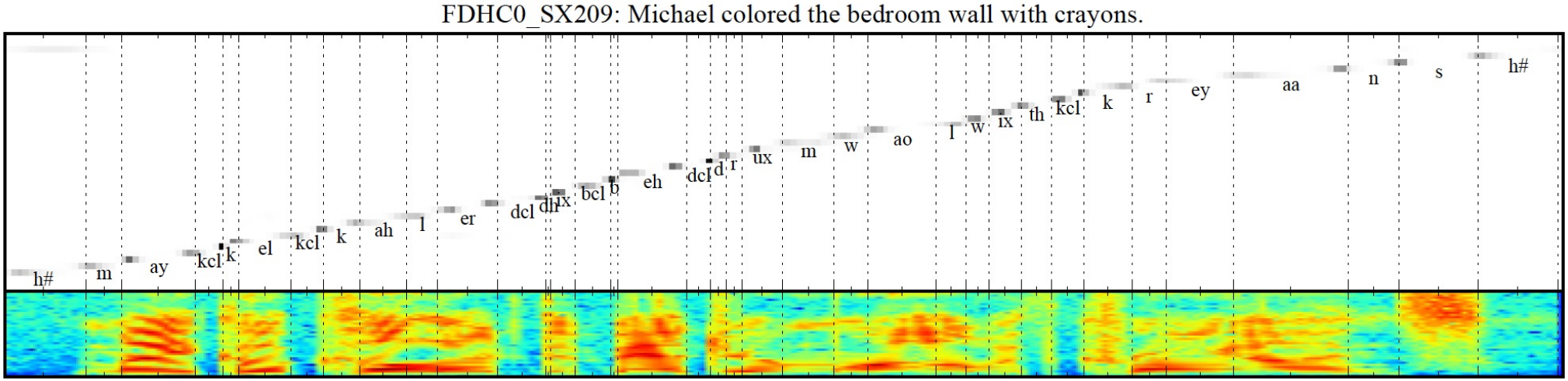
https://commons.wikimedia.org/wiki/File:Cocktail_Party_At_The_Imperial_Hotel_March_13,_1961_(Tokyo,_Japan)_(4966610682).jpg



https://commons.wikimedia.org/wiki/File:Cocktail-party_effect.svg

Bottom-up attention as a strategy for machine listening

- Suppose the most recent second of audio is stored in a “short-term memory buffer”
- A speech recognizer can attend to several centiseconds, all at one time, to decide what words it thinks it is hearing



Chorowski, Bahdanau, Serdyk, Cho & Bengio, Attention-Based Models for Speech Recognition, Fig. 1

Attention = a probability mass over time

- Attention is like probability: You only have a fixed amount of attention, so you need to decide how to distribute it.
- $\alpha_{i,t} = P(\boldsymbol{v}_t | q_i)$ = the probability that $\boldsymbol{v}_t$ is the context that you need in order to make a decision related to the query vector $\boldsymbol{q}_i$.

$$\sum_t \alpha_{i,t} = 1$$

- Each output context vector ($\boldsymbol{c}_i$) is based on some input value vectors ($\boldsymbol{v}_t$). But which ones? Answer: decide which inputs to pay attention to, then pay attention.

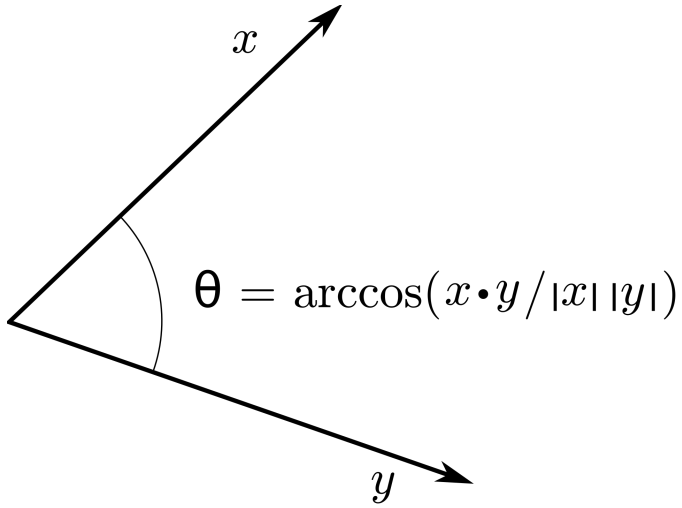
$$\boldsymbol{c}_i = \sum_t \alpha_{i,t} \boldsymbol{v}_t$$

Dot-product attention

How can you decide which value vectors, v_t are most relevant to a particular query? Answer:

1. Create a key vector, $\boldsymbol{k}_t$, such that $\boldsymbol{q}_i^T \boldsymbol{k}_t > 0$ if $\boldsymbol{v}_t$ is relevant to $\boldsymbol{q}_i$, otherwise $\boldsymbol{q}_i^T \boldsymbol{k}_t < 0$.
2. Convert the similarity measures into a probability distribution using softmax:

$$\alpha_{i,t} = \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)}$$



By BenFrantzDale at the English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=49972362

Putting it all together

- Stack up $\boldsymbol{v}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{q}_i$into matrices:

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_n^T \end{bmatrix}, \boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1^T \\ \vdots \\ \boldsymbol{k}_n^T \end{bmatrix}, \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_1^T \\ \vdots \\ \boldsymbol{q}_m^T \end{bmatrix}$$

- $\alpha_{i,t}$ is the t[th] output of a softmax whose input vector is $\boldsymbol{K}\boldsymbol{q}_i$:

$$\alpha_{i,t} = \text{softmax}_t(\boldsymbol{K}\boldsymbol{q}_i) = \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)}$$

- c_i is the product of the vector $\text{softmax}(\boldsymbol{K}\boldsymbol{q}_i)$ times the $\boldsymbol{V}^{\boldsymbol{T}}$ matrix:

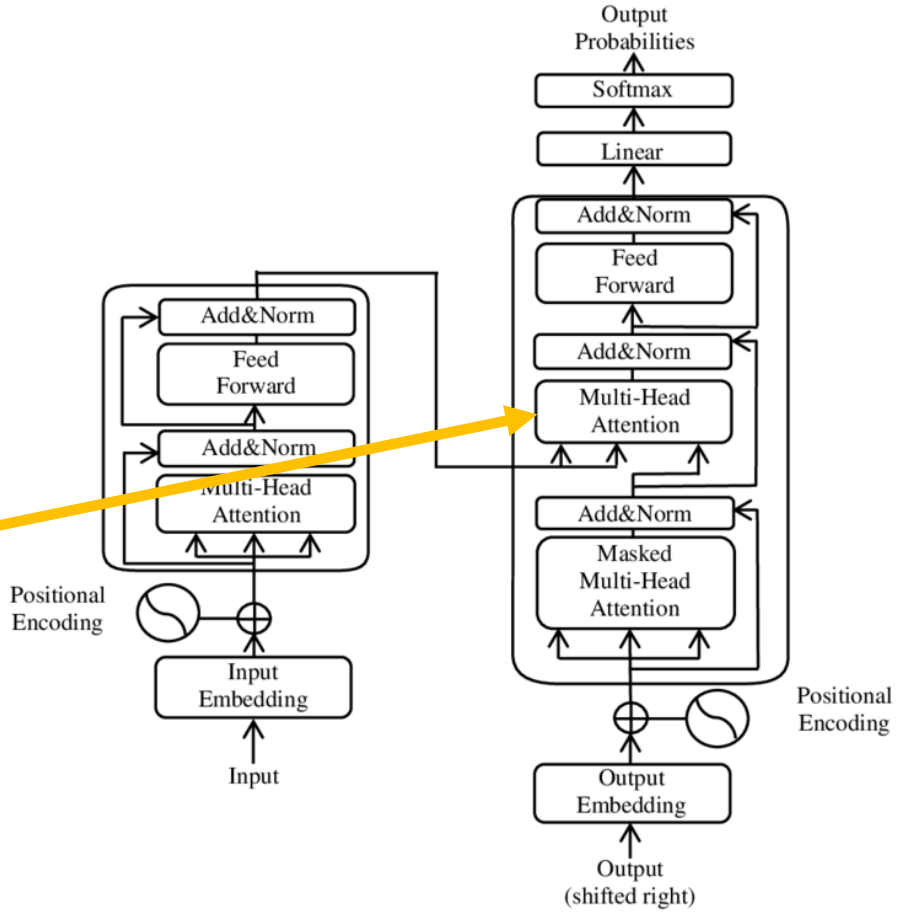
$$\boldsymbol{c}_i = \boldsymbol{V}^{\boldsymbol{T}} \text{softmax}(\boldsymbol{K}\boldsymbol{q}_i) = \sum_t \alpha_{i,t}\, \boldsymbol{v}_t$$

Attention

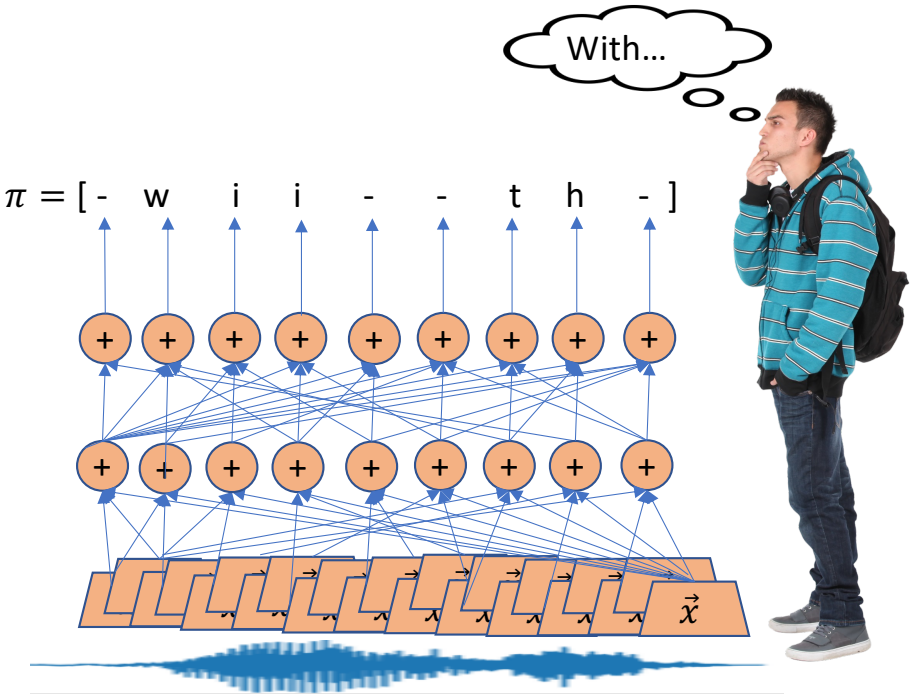
Query depends on preceding output, key and value depend on input:

$$\boldsymbol{q}_i = \boldsymbol{W}_Q \boldsymbol{o}_{i-1}$$
$$\boldsymbol{k}_t = \boldsymbol{W}_K \boldsymbol{x}_t$$
$$\boldsymbol{v}_t = \boldsymbol{W}_V \boldsymbol{x}_t$$

$$\boldsymbol{c}_i = \boldsymbol{V}^{\boldsymbol{T}} \mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_i) = \sum_t \alpha_{i,t} \boldsymbol{v}_t$$



Result: Process a sequence of vectors without any explicit concept of time



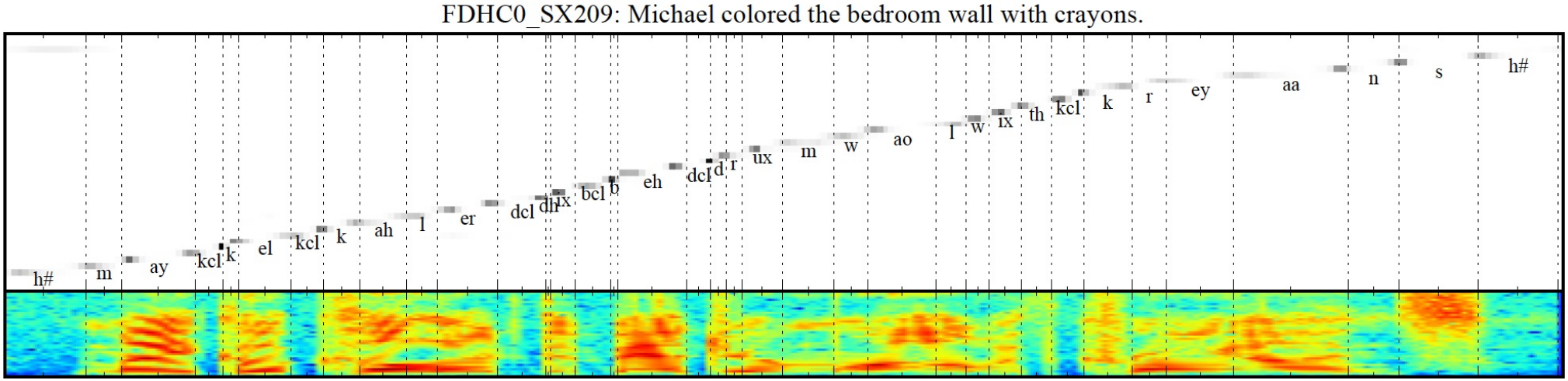
- Attention lets us process a sequence of vectors without knowing about time
- All we need to know: Find the vectors that are most like what we need next

Quiz!

Go to prairielearn and try the quiz!

Cross-attention visualization

This plot shows $\alpha_{i,t}$ where i = output character, and t = input spectrum



Chorowski, Bahdanau, Serdyk, Cho & Bengio, Attention-Based Models for Speech Recognition, Fig. 1

Outline

- Recurrent neural networks
- Attention
- Positional Encoding

What we have lost…

- With the recurrent neural net, each state vector paid attention to the one that preceded it:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{U}\boldsymbol{v}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$$

- With a transformer, each state vector pays attention to the input that is most similar, regardless of what time it happened:

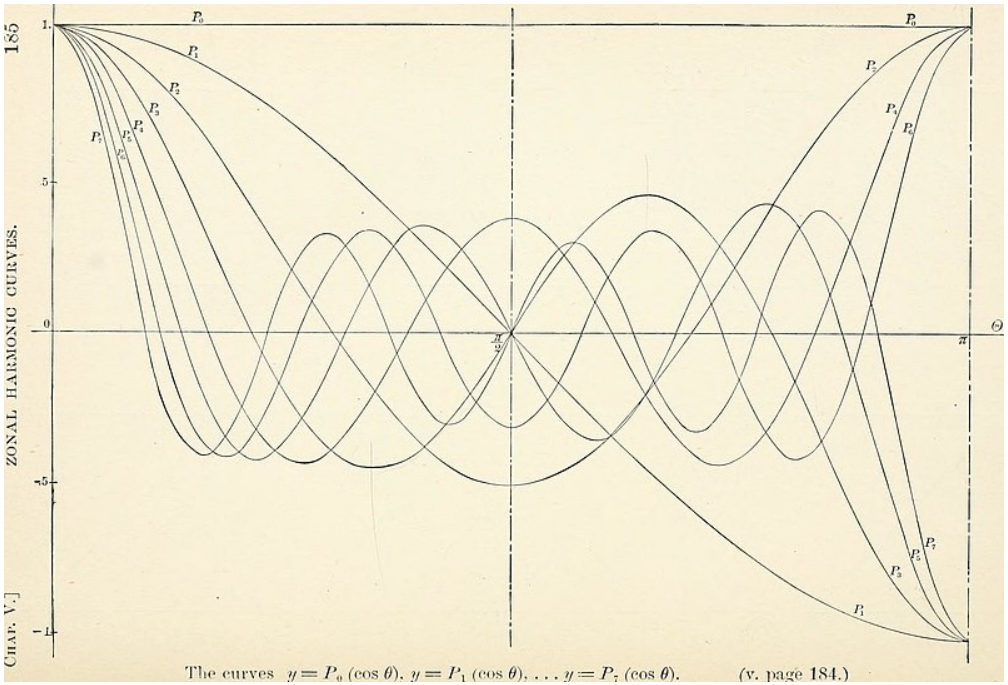
$$\boldsymbol{h}_i = \sum_t \alpha_{i,t}\, \boldsymbol{v}_t, \qquad \alpha_{i,t} = \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)}$$

- What if we always want $\boldsymbol{h}_t$ to pay special attention to $\boldsymbol{v}_{t-1}$? Is that possible?

Position encoding by Fourier basis

The solution is to encode the relative position of each input, $\boldsymbol{v}_t$, using a Fourier basis $\boldsymbol{e}_t$:

$$\boldsymbol{e}_t = \begin{bmatrix} \cos\left(\frac{\pi t}{T}\right) \\ \sin\left(\frac{\pi t}{T}\right) \\ \vdots \\ \cos\left(\frac{\pi D t}{2T}\right) \\ \sin\left(\frac{\pi D t}{2T}\right) \end{bmatrix}$$



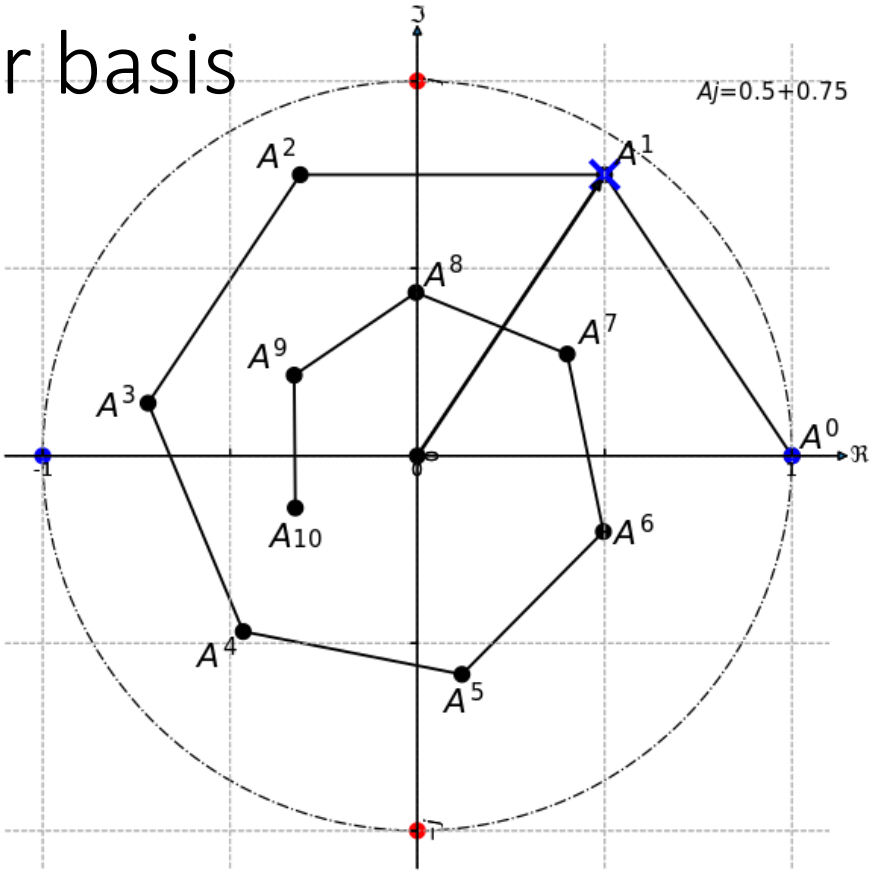
Public domain image, https://commons.wikimedia.org/wiki/File:An_elementary_treatise_on_Fourier%27s_series_and_spherical,_cylindrical,_and_ellipsoidal_harmonics,_with_applications_to_problems_in_mathematical_physics_(1893)_(14780364665).jpg

Position encoding by Fourier basis

The Fourier basis is useful because shifting by a fixed time offset, to $t - d$, can be accomplished by a matrix multiplication:

$$\boldsymbol{e}_{t-d} = \begin{bmatrix} \cos\left(\frac{\pi d}{T}\right) & \sin\left(\frac{\pi d}{T}\right) & \cdots \\ -\sin\left(\frac{\pi d}{T}\right) & \cos\left(\frac{\pi d}{T}\right) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \boldsymbol{e}_t$$

…so if we want a particular query to pay attention to vectors with a time delay of d, we just set $\boldsymbol{W}_{j,Q}$ to the matrix shown above.



Public domain image, https://commons.wikimedia.org/wiki/File:Exponentials_of_complex_number_within_unit_circle-2.svg

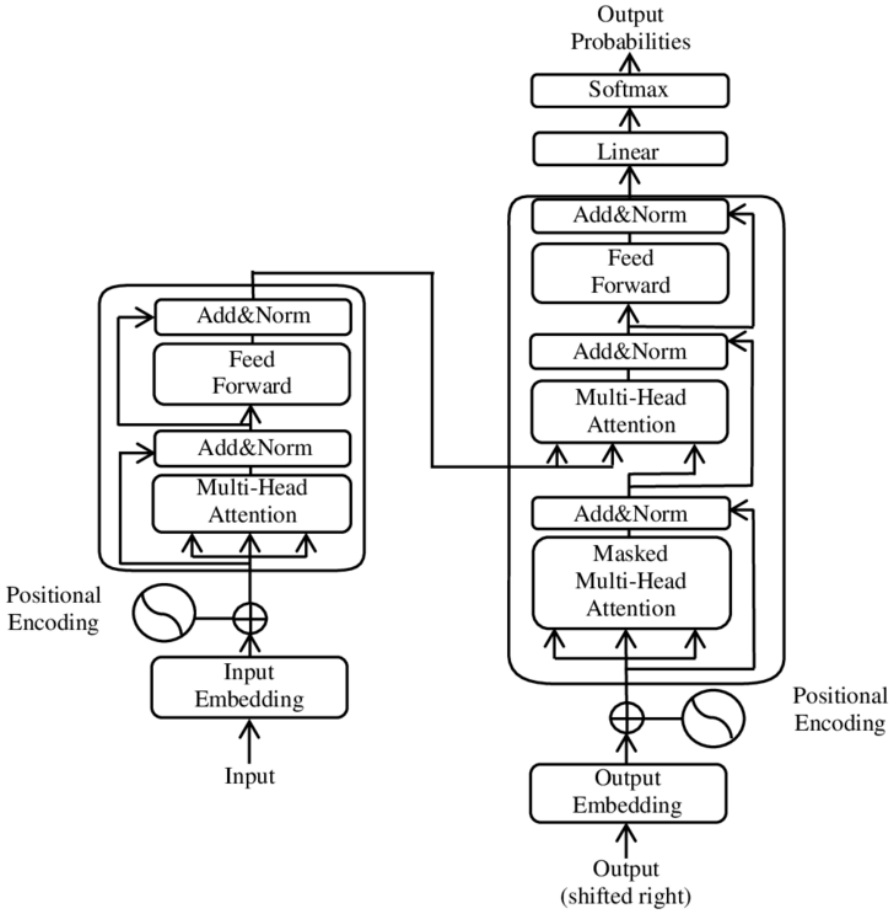
Where do we put the positional encoding?

- Possibility #1: Concatenate it, i.e., $\boldsymbol{x}_t^T = [\boldsymbol{x}_t^T, \boldsymbol{e}_t^T]$
  - Advantage: $\boldsymbol{W}_{j,Q}$ can learn to operate separately on the content $\boldsymbol{x}_t^T$ and the positional encoding $\boldsymbol{e}_t^T$
  - Disadvantage: every vector is twice as large, and every matrix is four times as large
- Possibility #2: Add it, i.e., $\boldsymbol{x}_t = \boldsymbol{x}_t + \boldsymbol{e}_t$
  - Advantage: fewer parameters to learn
  - Disadvantage: $\boldsymbol{W}_{j,Q}$ can only operate directly on $\boldsymbol{e}_t$ if $\boldsymbol{x}_t$ is mostly zero
  - Surprise: this works well in practice. Apparently, the positional encoding can learn to ignore local fluctuations in $\boldsymbol{x}_t$, and pretend that it's mostly 0 on average

Positional encoding

In the standard transformer, position of the input is encoded using

$$\boldsymbol{x}_t = \boldsymbol{x}_t + \begin{bmatrix} \cos\left(\frac{\pi t}{T}\right) \\ \sin\left(\frac{\pi t}{T}\right) \\ \vdots \\ \cos\left(\frac{\pi D t}{2T}\right) \\ \sin\left(\frac{\pi D t}{2T}\right) \end{bmatrix}$$



Summary

- Recurrent neural networks

$$\boldsymbol{h}_t = \tanh(\boldsymbol{U}\boldsymbol{v}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$$

- Attention

$$\boldsymbol{c}_i = \boldsymbol{V}^{\boldsymbol{T}}\mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_i) = \sum_t \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)} \boldsymbol{v}_t$$

- Positional encoding

$$\boldsymbol{x}_t \mathrel{+}= \begin{bmatrix} \cos\left(\frac{\pi t}{T}\right) \\ \sin\left(\frac{\pi t}{T}\right) \\ \vdots \end{bmatrix}$$